```
/*******************************************************************************
 Module
   GameFSM.c

 Revision
   1.0.1

 Description
   This is a template file for implementing flat state machines under the
   Gen2 Events and Services Framework.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 01/15/12 11:12 jec      revisions for Gen2 framework
 11/07/11 11:26 jec      made the queue static
 10/30/11 17:59 jec      fixed references to CurrentEvent in RunTemplateSM()
 10/23/11 18:20 jec      began conversion from SMTemplate.c (02/20/07 rev)
*******************************************************************************/
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "GameFSM.h"
#include <stdio.h>
#include <math.h>
#include "TargetControl.h" // Has all target helper functions
#include "BalanceFSM.h"
#include "TargetFSM.h"
#include "ES_PostList.h"
#include "VibrationControl.h"
#include "CommFSM.h"
#include "ServoLib.h"

#define SPACE_INDEX 47
/*----------------------------- Module Defines ----------------------------*/


#define READY 'p'

//#define KEYPRESS  1

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/

/*---------------------------- Module Variables ---------------------------*/
// everybody needs a state variable, you may need others as well.
```

```
// type of state variable should match htat of enum in header file
static GameState_t CurrentState;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitGameFSM

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, False if error in initialization, True otherwise

 Description
     Saves away the priority, sets up the initial transition and does any
     other required initialization for this state machine
 Notes

 Author
     J. Edward Carryer, 10/23/11, 18:55
****************************************************************************/
bool InitGameFSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the Initial PseudoState
  CurrentState = InitPState;

  Servo12_Init(SERVO_INIT_STRING);

  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService( MyPriority, ThisEvent) == True)
  {
      return True;
  }else
  {
      return False;
  }
}

/****************************************************************************
 Function
     PostGameFSM

 Parameters
     EF_Event ThisEvent , the event to post to the queue
```

```
 Returns
     bool False if the Enqueue operation failed, True otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostGameFSM( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunGameFSM

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   add your description here
 Notes
   uses nested switch/case to implement the machine.
 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event RunGameFSM( ES_Event ThisEvent )
{
  GameState_t NextState = CurrentState;

  ES_Event ReturnEvent;
  ES_Event NewEvent;
  ES_Event CommEvent;

  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  switch ( CurrentState )
  {
      // If current state is initial Psedudo State
   case InitPState :
          if ( ThisEvent.EventType == ES_INIT )// only respond to ES_Init
          {
              puts("Power Up - GAME B \r");

              //Turn servo to 0
               Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);
```

```
            // Move into Lockdown
            NextState = Lockdown;
            puts("Move to: Lockdown \r");
        }
        break;

//*****************************LOCKDOWN*********************************//
//State: Lockdown (No Access Card)
case (Lockdown):
        puts("In Lockdown \r");
        // If access card has been detected
        if (ThisEvent.EventType == ES_AccessCardDetected ){               //Internal event
        from comm
            puts("Access Card Detected. Move to: WaitForReady \r");

            //Turn on all LEDs
            RunWaitingForStartShow();

            //Init Balance Board and Target - necessary??
            NewEvent.EventType = ES_INIT;
            ES_PostList01(NewEvent);

            //Turn servo to 0
            Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);

            NextState = WaitForReady;
        }



        break;

//*****************************READY BUTTON*********************************//
//State: ReadyButton -> Wait for Both
 case(WaitForReady):
        if(ThisEvent.EventType == ES_AccessCardRemoved){                  //Internal event
        from comm

                puts("Access card removed. Move to: Lockdown \r");

                NextState = Lockdown;

                TurnOffAllTargets();
                TurnVibrationOff();
                StopShow();

                //Turn servo to 0
                Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);

                break;
        }
```

```
            //If B's button has been pressed
            #ifdef KEYPRESS
            else if(ThisEvent.EventType == ES_NEW_KEY && ThisEvent.EventParam == READY){
            #endif
            #ifndef KEYPRESS
            else if(ThisEvent.EventType == ES_ButtonDown){//ES_NEW_KEY):
            #endif

                    //if(ThisEvent.EventParam == READY){
                        puts("B button press. Move to: BReady \r");

                        // Change states
                        NextState = BReady;

                        CommEvent.EventType =  ES_SEND_REQUEST;
                        CommEvent.EventParam = BReadyPushMessage;
                        PostCommFSM(CommEvent);

                        StopShow();
                        TurnOffAllTargets();
                    //}

            }

            break;



    //State: Ready Button -> B is Ready, Waiting for A
     case (BReady):
            if (ThisEvent.EventType == ES_AccessCardRemoved){                //Internal event
            from comm
                NextState = Lockdown;

                //Turn off all LEDs
                TurnOffAllTargets();
                StopShow();

                //Vibration off
                TurnVibrationOff();

                puts("Access card removed. Move to: Lockdown \r");

            }else if (ThisEvent.EventType == ES_StageEnd) {          //Internal event from comm

                StartPracticeRound();

                //Post Begin event to Target Interpreter  &Balance
                NewEvent.EventType = ES_Begin;
                ES_PostList01(NewEvent);

                NextState = WaitTargetPractice;
```

```
            puts("Both ready. Move to: WaitTargetPractice \r");




        }

        break;

//*************************TARGET PRACTICE*********************************//
//State: Target Practice
 case (WaitTargetPractice):
        if (ThisEvent.EventType == ES_AccessCardRemoved ){                //Internal event
        from comm
            NextState = Lockdown;

            //Turn off all LEDs
            TurnOffAllTargets();

            puts("Access card removed. Move to: Lockdown \r");

            //Turn servo to 0
             Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);

            break;

        }else if (ThisEvent.EventType == ES_StageEnd){                //Internal event from
        comm
            //BalanceState_t BoardState = QueryBalanceFSM();

                puts("Target Practice Complete. Move to: Balance Practice \r");

            TurnOffAllTargets();
            DimTarget();
            BlinkTarget();

            //Vibration on
            TurnVibrationOn();
            NextState = BalancedBPractice;


        }

        else if (ThisEvent.EventType == ES_GameEnd){                    //Internal event
        from comm
            NextState = Celebration;
            RunEndOfGameShow();

            //Turn servo to 0
           Servo12_SetPulseWidth(SERVO_PIN,  SERVO_90_DEG);

            puts("Game timeout. Move to: Celebration \r");
        }
```

```
        break;

//************************BALANCE PRACTICE*************************************//
//State: Balance Practice -> Balanced
 case (BalancedBPractice):
        if(ThisEvent.EventType == ES_AccessCardRemoved){         //Internal event from comm

                NextState = Lockdown;

                //Turn off all LEDs
                TurnOffAllTargets();

                //Vibration off
                TurnVibrationOff();

                //Turn servo to 0
           Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);

                puts("Access Card Removed. Move to:Lockdown \r");

                break;

        } else if (ThisEvent.EventType == ES_StageEnd){ //ES_NEW_KEY):      //Internal
        event from comm
                //Balance Timeout

                    NextState = WaitingTargetSynchro;
                    StopShow();
                    StartSynchroRound();

                    //Post Balance Complete message to BalanceFSM
                    NewEvent.EventType = ES_BalanceComplete;
                    PostBalanceFSM(NewEvent);

                    //Post begin to target FSM
                    NewEvent.EventType = ES_Begin;
                    PostTargetFSM(NewEvent);

                    TurnVibrationOff();
                    puts("Balance Practice Complete. Move to: Wait Target Synchro \r");

                break;

                //Game Timeout
        } else if (ThisEvent.EventType == ES_GameEnd){                    //Internal event
        from comm
                    NextState = Celebration;
                    //vibration off
                    TurnVibrationOff();
                    RunEndOfGameShow();
                    //move servo
                    //Turn servo to 90
            Servo12_SetPulseWidth(SERVO_PIN,  SERVO_90_DEG);
```

```
                    puts("Game Timeout. Move to: Celebration \r");

        }
        break;



//**********************************SYNCHRO*************************************//
//State: Synchro -> Wait for Target
 case (WaitingTargetSynchro):

        if(ThisEvent.EventType == ES_AccessCardRemoved){
                NextState = Lockdown;                                   //Internal
                event from comm
                //Turn off all LEDs
            TurnOffAllTargets();

            puts("B: Access Card Removed. Move to: Lockdown \r");
        }

        //If they both cover the targets or a timeout happens
        else if (ThisEvent.EventType == ES_AdvanceTarget){
                //AdvanceTarget                                          //Internal
                event from comm
                AdvanceTarget();

            puts("Target Hit!! Stay in Waiting Target Synchro \r");
        }

        else if (ThisEvent.EventType == ES_GameEnd){                    //Internal
        event from comm

                NextState = Celebration;
                RunEndOfGameShow();
                //move servo
                //Turn servo to 0
           Servo12_SetPulseWidth(SERVO_PIN,  SERVO_90_DEG);
                puts("Game Timeout. Move to: Celebration \r");
        }

        break;



//*********************************CELEBRATION**************************************//

// State: Celebrate!! (End of Game)
 case (Celebration):
        puts("CELEBRATE!!!");

        if (ThisEvent.EventType == ES_AccessCardRemoved){               //Internal
        event from comm
```

```
                NextState = Lockdown;

                //turn off all leds
                TurnOffAllTargets();
                StopShow();

                //Turn servo to 0
               Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);

                puts("B: Access Card Removed. Move to: Lockdown \r");
            }

            else if (ThisEvent.EventType == ES_StageEnd){                          //Internal
            event from comm
                //Turn on all LEDs
                StopShow();
                RunWaitingForStartShow();

                NextState = WaitForReady;

                //Turn servo to 0
            Servo12_SetPulseWidth(SERVO_PIN,  SERVO_0_DEG);

                puts("B: Power Down Timeout. Move to: Wait For Ready \r");
            }

          break;

  }

  CurrentState = NextState;

  return ReturnEvent;
}

/****************************************************************************
 Function
     QueryTemplateSM

 Parameters
     None

 Returns
     TemplateState_t The current state of the Template state machine

 Description
     returns the current state of the Template state machine
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:21
****************************************************************************/
GameState_t QueryGameFSM ( void )
```

```
{
   return(CurrentState);
}

/*****************************************************************************
 private functions
 *****************************************************************************/
```